Algorithmica (1998) 20: 136–150

Algorithmica

Efficient Randomized Algorithms for the Repeated Median Line Estimator[1]

J. Matoušek,[2] D. M. Mount,[3] and N. S. Netanyahu[4]

Abstract. The problem of fitting a straight line to a finite collection of points in the plane is an important problem in statistical estimation. Recently there has been a great deal of interest is *robust estimators*, because of their lack of sensitivity to outlying data points. The basic measure of the robustness of an estimator is its *breakdown point*, that is, the fraction (up to 50%) of outlying data points that can corrupt the estimator. One problem with robust estimators is that achieving high breakdown points (near 50%) has proved to be computationally demanding. In this paper we present the best known theoretical algorithm and a practical subquadratic algorithm for computing a 50% breakdown point line estimator, the Siegel or repeated median line estimator. We first present an $O(n \log n)$ randomized expected-time algorithm, where n is the number of given points. This algorithm relies, however, on sophisticated data structures. We also present a very simple $O(n \log^2 n)$ randomized algorithm for this problem, which uses no complex data structures. We provide empirical evidence that, for many realistic input distributions, the running time of this second algorithm is actually $O(n \log n)$ expected time.

Key Words. Repeated median estimator, Line fitting, Robust estimators, Randomized algorithms, Computational geometry.

1. Introduction. Fitting a straight line to a finite collection of data points in the plane is a fundamental problem in statistical estimation, with numerous applications. Although methods such as least squares are well understood and easy to compute, these methods are known to suffer from the phenomenon that a small number of outlying points can perturb the line of fit by an arbitrarily large amount. For this reason, there has been increased interest in a class of estimators, called *robust estimators* [17], [14], [27], that do not suffer from this deficiency. Define the *breakdown point* of an estimator to be the fraction of outlying data points (up to 50%) that may cause the estimator to take on an arbitrarily large aberrant value. (See [8] and [27] for exact definitions.) The breakdown point of an estimator is a measure of its robustness. For example, the (asymptotic) breakdown point

[1] A preliminary version of this paper was presented at the *Fourth Annual ACM–SIAM Sympsium on Discrete Algorithms* [22]. Jiří Matoušek has been supported by a Humboldt Research Fellowship. David Mount has been supported by National Science Foundation Grants CCR-89-08901 and CCR-93-10705. Nathan Netanyahu was supported by the Air Force Office of Scientific Research under Grant AFOSR-91-0239 (while a doctoral candidate at the University of Maryland), and has completed this research while holding a National Research Council NASA Goddard Associateship.

[2] Department of Applied Mathematics, Charles University, Prague, Czech Republic.

[3] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA.

[4] Center for Automation Research, University of Maryland, College Park, MD 20742, USA and Center of Excellence in Space Data and Information Sciences, Code 930.5, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA.

Received January 25, 1995; revised May 17, 1996. Communciated by L. J. Guibas.

of least squares is zero because even a single outlying data point can have an arbitrarily large effect on the estimator. Examples of robust estimators include the following:

Theil–Sen estimator: The slope of the line of fit is taken to be the median of the set of $\binom{n}{2}$ slopes[5] that result by passing a line through each pair of distinct points in the data set [34], [29]. In the plane, the Theil–Sen estimator has a breakdown point of $\approx 29.3\%$.

This problem has been studied under the name of the *slope selection problem* in the field of computational geometry. The problem is to determine the slope of any given rank. There exist asymptotically optimal algorithms for this problem, which run in $O(n \log n)$ time and $O(n)$ space. These include algorithms by Cole *et al*. [5], Katz and Sharir [18], and Brönnimann and Chazelle [1].

It should be noted that all of the above algorithms rely on fairly complicated techniques. (Chazelle *et al*. [2] have presented a simpler deterministic algorithm, but its running time is $O(n \log^2 n)$.) There are simpler, practical *randomized* algorithms by Matoušek [20] and Dillencourt *et al*. [7], and Shafer and Steiger [30]. These are *Las Vegas* randomized algorithms, meaning that they always produce correct results, and on any input, the expected running time, when averaged over the random choices made in the algorithm, is $O(n \log n)$.

LMS estimator: Rousseeuw's *least median of squares* (LMS) *estimator* [26] is defined to be the line that minimizes the median of the squared residuals. LMS has a breakdown point of 50%. The best known algorithms for LMS run in $O(n^2)$ time, due to Souvaine and Steele [32] and Edelsbrunner and Souvaine [9].

RM estimator: Siegel's *repeated median* (RM) *estimator* [31] of a set of n distinct points in the plane $\{p_1, p_2, \ldots, p_n\}$ is defined as follows. For each point $p_i = (x_i, y_i)$, let θ_i denote the median of the $n-1$ slopes of the lines passing through p_i and each other point of the set. The *RM-slope*, θ^*, is defined to be the median of the multiset $\{\theta_i\}$.[6] The (hierarchical) *RM-intercept* is defined, in terms of the computed repeated median slope, to be the median of the multiset, $\{y_i - \theta^* x_i\}$. The repeated median estimator has a breakdown point of 50%.

Observe that once the RM-slope has been computed, the RM-intercept can be computed easily in $O(n)$ additional time by any linear-time selection algorithm. Thus we concentrate on the task of computing the RM-slope. A brute-force algorithm for computing the RM-slope takes $O(n^2)$ time and $O(n)$ space. Recently Stein and Werman [33] proposed a deterministic algorithm for the RM-slope which runs in $O(n \log^2 n)$ time. However, their algorithm is rather complex, relying on several involved methods, such as Cole's improvement [4] to Megiddo's parametric search technique [23].

In this paper we present two Las Vegas randomized algorithms for computing the repeated median slope. Both algorithms are conceptually very simple, and use linear storage. The first runs in $O(n \log n)$ expected time, improving Stein and Werman's result. This algorithm is optimal for the generalization to arbitrary selection mentioned in the footnote, since element uniqueness can easily be reduced to this problem. The

[5] For the purposes of this paper we define the *median* of an m element multiset to be an element of rank $\lceil m/2 \rceil$.
[6] The above definitions can be generalized to the selection of elements of *arbitrary* rank either for the choice of the θ_i's, or for the choice of θ^*.

algorithm relies, however, on data structures for half-plane range searching, which are fairly sophisticated. The second algorithm runs in $O(n \log^2 n)$ expected time ($O(\log n)$ iterations of a procedure that runs in $O(n \log n)$ time). This algorithm uses no sophisticated data structures (only arrays), is quite easy to implement, and has a good running time. Furthermore, we provide empirical evidence that for many typical input instances the expected running time of this algorithm is $O(n \log n)$ (because the number of iterations is constant).

In Section 2 we describe elements common to the two approaches and focus, in particular, on the $O(n \log n)$ algorithm, which is based on range searching. In Section 3 we present a practical algorithm, which is based on Monte Carlo median estimates. Section 4 provides experimental results of the performance of the latter algorithm. Section 5 contains concluding remarks.

2. The Algorithm. In this section we present the algorithmic framework common to both algorithms and describe, specifically, the $O(n \log n)$ expected-time randomized algorithm. Before introducing the algorithmic framework, it is conceptually helpful to map the problem into its dual form. Recall that the input is a collection of n points $p_i = (x_i, y_i)$, $i = 1, \ldots, n$. We map each point (x, y) from the original problem into a line in dual space. Let θ and φ denote the coordinates of dual space. The point (x, y) is mapped into the dual line

$$\{(\theta, \varphi) \mid \varphi = x\theta - y\}.$$

It is easy to verify that given two distinct points in primal space, the θ-coordinate of the intersection of their two dual lines is the slope of the line in primal space passing through these points. (As in [7], we make the conventions that a vertical line has a slope of $+\infty$ and parallel lines meet at $+\infty$.) The θ-coordinate of the intersection of two dual lines is called an *intersection abscissa*. By the *intersection abscissas* of a given dual line, we mean the $n - 1$ intersection abscissas formed between this line and all the other dual lines. Thus, given a point $p_i = (x_i, y_i)$, the slopes of the $n - 1$ lines formed by joining p_i and each of the remaining points of the set are just the intersection abscissas of the dual of p_i.

By considering the arrangement of the n dual lines in (θ, φ) space, we can pose the repeated median slope problem in the following equivalent form. For the dual line of point p_i, consider its median intersection abscissa, θ_i. The RM-slope θ^* is the median of these medians. Figure 1(a) illustrates a line arrangement in dual space where $n = 4$. The median θ-coordinates of the intersections lying on the dual lines l_1, l_2, l_3, and l_4 are θ_2, θ_5, θ_4, and θ_4, respectively. The repeated median is the median of these abscissas, namely θ_4.

To simplify the presentation, we make the general position assumption that no two pairs of points generate the same slope, which further implies that no three points are collinear. (A more detailed report describes a complete implementation of the second algorithm, which allows arbitrary point placement [24].)

We now digress momentarily to present the basic probability theoretic construct on which our methodology relies. The lemma follows from Lemmas 3.1 and 3.2 in [7], but

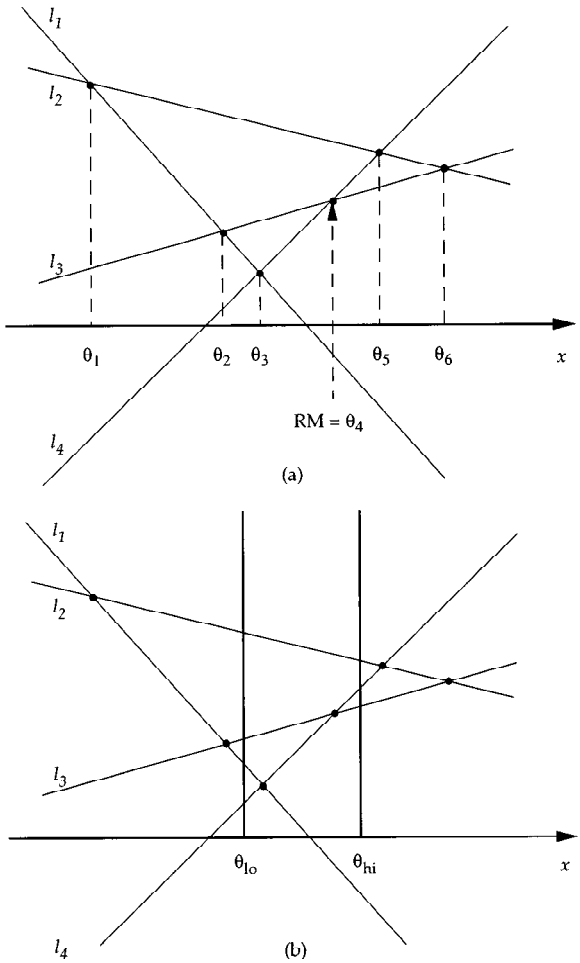

Fig. 1. Finding the RM in dual space: (a) An example of a line arrangement in dual space, for $n = 4$. The RM is the abscissa θ_4. (b) The sets L, C, and R with respect to the depicted interval, $(\theta_{lo}, \theta_{hi}]$: $L = \{l_1\}$, $C = \{l_3, l_4\}$, $R = \{l_2\}$.

we provide a detailed sketch here for completeness. Intuitively it states that given a set of n numbers from which we can sample at random, we can compute a small confidence interval for the kth smallest member of the set in time that is essentially independent of n. The running time of the procedure, the degree of confidence, and the size of the interval are related to one another.

LEMMA 2.1. *Given a set of numbers $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$, an index k $(1 \leq k \leq n)$, and an integer $r > 0$, we can compute in $O(r)$ time an interval $[\theta_{lo}, \theta_{hi}]$, such that, with probability $1 - 1/\Omega(\sqrt{r})$, the kth smallest element of Θ lies within this interval, and the number of elements in Θ that lie within the interval is at most $n/\Omega(\sqrt{r})$.*

PROOF: Sample (with replacement) r of the elements of Θ, and using any fast selection algorithm, select from this sample the elements θ_{lo} and θ_{hi} whose respective ranks are

$$k_{\text{lo}} = \max\left(1, \left\lfloor \frac{rk}{n} - \frac{3\sqrt{r}}{2} \right\rfloor\right),$$
$$k_{\text{hi}} = \min\left(r, \left\lceil \frac{rk}{n} + \frac{3\sqrt{r}}{2} \right\rceil\right).$$

This can be done in $O(r)$ time. (For our purposes, it suffices to use a simpler *randomized* algorithm for selection (see, e.g., [16], [13], and [6] for specific details).)

The kth smallest element is less than θ_{lo} if and only if fewer than k_{lo} sampled elements are less than the kth smallest element. Since the probability that a given element is less than or equal to the kth smallest element is k/n, it follows that, in r samples, the number of sampled values that are less than the kth smallest element is a binomial random variable with mean rk/n and standard deviation not greater than $\sqrt{r}/2$. The probability that fewer than k_{lo} sampled elements are less than θ_{lo} is essentially the probability that the binomial random variable is at least three standard deviations below its mean value. By applying Chernoff's bounds (see, e.g., [3] and [11]) and Chebyshev's inequality [12], it follows that this probability is $1/\Omega(\sqrt{r})$. (See Lemmas 3.1 and 3.2 in [7] for complete details.) A similar argument applies for k_{hi}, and the probability that the kth smallest element does lie within the interval is, therefore, $1 - 1/\Omega(\sqrt{r})$.

Since a fraction of the sample of size $O(\sqrt{r})/r$ lies within the interval (by definition), it follows that the expected fraction of Θ that lies within the interval is $nO(\sqrt{r})/r = n/\Omega(\sqrt{r})$. Again, Chernoff's bounds can be invoked to show that this occurs with at least the stated probability. □

We return to the description of the algorithm. We apply an interval contraction technique, which is quite similar to the one used in the randomized algorithms for the Theil–Sen estimator [7], [20]. We maintain an interval $(\theta_{\text{lo}}, \theta_{\text{hi}}]$ that contains the repeated median slope. The initial interval is $(-\infty, +\infty]$. (This is consistent with the conventions that were adopted from [7], namely that an interval is treated as half-open half-closed, a vertical line has a slope of $+\infty$, and no two lines intersect at $-\infty$.) The interval is contracted through a series of stages. During each stage we construct a subinterval $(\theta'_{\text{lo}}, \theta'_{\text{hi}}]$ that contains the repeated median. It will be shown that each stage runs in $O(n \log n)$ expected time, and the number of stages in the expected case is $O(1)$.

We consider the operation of a stage in greater detail. For the dual line of point p_i, $i = 1, \ldots, n$, we maintain three counts: L_i, R_i, and C_i, which denote, respectively, the number of the line's intersection abscissas that lie to the left, to the right, and within the interval. (We discuss later how these counts are computed.) Depending on the relationship between $\lceil (n-1)/2 \rceil$, L_i, and $L_i + C_i$, we can determine whether the line's median intersection abscissa lies to the left, to the right, or within the interval. We partition the set of lines into three subsets L, R, and C, accordingly.

For example, in Figure 1(b) we have $L = \{l_1\}$, $C = \{l_3, l_4\}$, and $R = \{l_2\}$. Since we assume that the repeated median lies within C, it follows that $|L| < \lceil n/2 \rceil \leq |L| + |C|$. A dual line is a *candidate* to supply the final repeated median if it lies in C. In particular,

the candidate whose median intersection abscissa is of rank $\lceil n/2 \rceil - |L|$ within C is the desired candidate.

In order to contract the interval, we apply Lemma 2.1 to the set of median intersection abscissas of the candidate lines. Since we do not have ready access to this set, the procedures used to access elements of this set randomly are discussed later. Assuming that this subproblem can be handled, Lemma 2.1 suggests the following general algorithm:

(1) Set the initial interval to $(-\infty, +\infty]$. Initialize counts, such that $L_i := R_i := 0$ and $C_i := n-1$. Likewise, initialize sets, such that $L := R := \emptyset$ and $C := \{1, 2, \ldots, n\}$.
(2) Repeat the following steps until $|C| = 1$ (or, more practically, until $\sum_{i \in C} C_i = O(n)$, after which brute-force enumeration can be used).
  (2a) Let β $(0 < \beta < 1)$ be a constant whose value is considered later. Set $r := \lceil n^{\beta} \rceil$, and sample r dual lines of C randomly, with replacement.
  (2b) For each sampled line, compute its median intersection abscissa (by a method to be described later).
  (2c) Let $k := \lceil n/2 \rceil - |L|$ (the rank of the repeated median in C), and, applying Lemma 2.1, let

$$k_{\text{lo}} := \max\left(1, \left\lfloor \frac{rk}{|C|} - \frac{3\sqrt{r}}{2} \right\rfloor\right),$$
$$k_{\text{hi}} := \min\left(r, \left\lfloor \frac{rk}{|C|} + \frac{3\sqrt{r}}{2} \right\rfloor\right).$$

  Employ any fast selection algorithm to determine the elements θ'_{lo} and θ'_{hi} of these respective ranks from the sampled median intersection abscissas.
  (2d) For each dual line in C count the number of intersection abscissas that lie in each of the intervals $(\theta_{\text{lo}}, \theta'_{\text{lo}}]$, $(\theta'_{\text{lo}}, \theta'_{\text{hi}}]$, and $(\theta'_{\text{hi}}, \theta_{\text{hi}}]$. (Since we treat each interval as if it is open on the left and closed on the right, no intersection abscissa lies in more than one interval. Note also that only two of the counts need to be considered, since the third count can be inferred from the other two.)
  (2e) Based on these counts, determine which of the subintervals contains the repeated median and contract to this subinterval. (We expect it to be the middle subinterval, with high probability.) Update the counts L_i, R_i, and C_i, and the sets L, R, and C, accordingly.

To derive the running time of the algorithm, we must consider a number of issues in greater detail. The first issue is the number of iterations required until the algorithm terminates. Lemma 2.1 states that, with high probability, that is with probability greater than or equal to $1 - 1/\Omega(n^{\beta/2})$, the repeated median is trapped within the interval $(\theta'_{\text{lo}}, \theta'_{\text{hi}}]$ and the size of C is reduced by a fraction of $1/\Omega(n^{\beta/2})$. After repeating this process t times, the number of candidate lines will decrease by a factor of $1/\Omega(n^{t\beta/2})$. Since we began with n candidates, after a constant number of t stages ($t \in O(2/\beta)$), we will have found the repeated median. Thus it suffices to show that each stage can be performed in $O(n \log n)$ expected time.

It is easy to verify that each of the steps of the algorithm can be performed in $O(n)$ time, except for the operations of computing the counts of the number of intersection

abscissas that lie within a given interval, and computing the median intersection abscissa on each of the sampled lines. We address these issues in the next two subsections.

2.1. *Counting Intersection Abscissas.* We consider the problem of computing the number of intersection abscissas, L_i, R_i, and C_i, that lie, respectively, to the left, to the right, and within an interval, say $(\theta_{\text{lo}}, \theta_{\text{hi}}]$. Observe that, for each line, the sum of these counts is $n-1$, and, hence, over all lines it is quadratic in n. Thus we cannot simply compute these counts naively. We apply a standard reduction to the problem of counting the inversions in a permutation. An *inversion* in a list of numbers is defined to be a pair of elements where the smaller element appears later in the list than the larger element. We present a simple variant of the inversion-counting algorithm discussed in [7], where inversions are counted individually for each line. First we sort and label the dual lines according to their intersections with the vertical line $\theta = \theta_{\text{lo}}$, and then we consider the permutation of this labeling that results when we look at the intersections of the dual lines with the vertical line $\theta = \theta_{\text{hi}}$. It is easy to see that the inversions in this permutation are in one-to-one correspondence with the pairs of dual lines intersecting each other within the interval (because their relative order of intersection with the vertical lines has changed).

It is a simple exercise to modify mergesort to count the number of inversions in which each element is involved. Recall that mergesort [19] operates by splitting a list into left and right sublists (of roughly equal sizes) which are sorted recursively and then merged into a single sorted list. In order to count inversions, each element maintains a count of the number of elements with which it is involved in an inversion. When the recursive sorting calls are made, we count the number of inversions (per element) generated within each of the sublists. When two sublists are merged, each element from, say, the left sublist implicitly discovers the number of elements from the right sublist that are smaller. This number is added to the inversion counter for the element. A symmetric argument applies to elements of the right sublist. Thus all inversions can be counted in $O(n \log n)$ time and $O(n)$ space.

2.2. *Median Computation.* In this subsection we show how to compute the median intersection abscissa for each of the n^β sampled lines in (overall) $O(n \log n)$ expected time. Our approach is based on existing algorithms for solving range queries. Define a *double wedge* as the symmetric difference of two half-planes, that is, the locus of points that lie in an infinite bow-tie-shaped region between two lines. We make use of the following lemma, which follows easily from known results in range searching.

LEMMA 2.2. *Given a collection P of n points in the plane, they can be preprocessed in $O(n \log n)$ time and $O(n)$ space, so that given a query double wedge w, the number of points of $P \cap w$ can be counted in $O(n^\gamma)$ time, where γ is a constant smaller than one. Furthermore, a point from $P \cap w$ can be sampled at random in $O(n^\gamma)$ time.*

PROOF: The first statement, concerning the counting queries, is a well-known result from range searching (see, e.g., [35], [10], [15], and [21] for various solutions). In order to sample a random point, we can apply a standard trick to convert a counting procedure into a sampling procedure. We look more closely at the range-counting algorithms. Given a query double wedge w, they operate by decomposing the set $P \cap w$ into a collection

of $O(n^\gamma)$ disjoint subsets. These subsets arise from a fixed collection of subsets of P (so-called *canonical subsets*), which are (implicitly) represented in the range-counting data structure. Specifically, the data structure stores the cardinality of each canonical subset.

For a query double wedge w, we identify the $m = O(n^\gamma)$ canonical subsets $C_1, \ldots, C_m$, such that $P \cap w$ is the disjoint union of $C_1, \ldots, C_m$. (These sets are listed in some arbitrary but fixed order.) Given that w contains k points, we generate a random integer r in the range from 1 to k, and find the index j, such that $|C_1| + \cdots + |C_{j-1}| < r \leq |C_1| + \cdots + |C_{j-1}| + |C_j|$. This can easily be done since we know the cardinalities of the canonical subsets. It remains, therefore, to draw the $(r - |C_1| - \cdots - |C_{j-1}|)$th point of C_j, again, in accordance with some arbitrary but fixed ordering of the points in C_j.

Note that the sum of the cardinalities of all the canonical subsets is $O(n \log n)$. Hence, in $O(n \log n)$ space, we can simply store the elements of each canonical subset in an array, and then perform point selection from a given canonical subset. Assuming, however, that query processing involves searching a partition tree (of any of the types discussed in the above-mentioned range-searching references), the space can be reduced to $O(n)$. This is possible since all the canonical subsets are represented implicitly by a rooted tree, and point selection from a given canonical subset can be performed by tracing the appropriate branch in the tree until reaching the desired leaf. A reader familiar with the data structure(s) can easily complete the algorithm. We omit a detailed discussion, since it would require a specific description of the range-counting data structure. (Again, see the references mentioned at the beginning of the proof for further details.) □

We now define the constant β introduced in step (2) of the algorithm to be $1 - \gamma$ from the previous lemma. Using this relation, we show how to compute the median intersection abscissas of all the sampled dual lines in expected time $O(n \log n)$. For each of the n^β sampled lines, we describe a randomized binary search to compute the median intersection abscissa, where each probe of the binary search will take $O(n^\gamma)$ time. The expected number of probes will be $O(\log n)$, yielding, thereby, a total running time of $O(n^\beta n^\gamma \log n) = O(n \log n)$.

To describe the binary search, we return to primal space. Let p_i be a point whose dual line was sampled. Recall that we want to compute the median of the slopes of the $n - 1$ lines passing through p_i and each other point of the data set. A general step of the binary search works as follows. We assume that we have confined this median slope to lie within some interval $(\theta_1, \theta_2]$. A point p_j $(i \neq j)$ defines a line $p_i p_j$ whose slope lies within this range if and only if p_j lies within a wedge centered at p_i and whose lines have slopes θ_1 and θ_2. By Lemma 2.2, we can sample at random one of the points from this wedge, say p_j, in $O(n^\gamma)$ expected time. Let θ' be the slope of the line $p_i p_j$. Thus, by applying once again the previous lemma, we can count the number of points in the wedge $(\theta_1, \theta']$, i.e., the number of intersection abscissas in this subinterval, in $O(n^\gamma)$ time. This will enable us to determine whether the median intersection abscissa lies within this subinterval or within the subinterval, $(\theta', \theta_2]$, and recurse on the appropriate one. Since the probe point is chosen randomly from within the range, the expected number of points eliminated by each probe is a constant fraction of the remaining points. Thus, there are $O(\log n)$ probes in the expected case.

3. Estimating Median Intersection Abscissas. Although the algorithm presented in the previous section is asymptotically efficient, its reliance on data structures for half-plane range queries makes is less attractive for practical implementation. In this section we present a more practical approach, which replaces step (2b) of the previous algorithm by a scheme that *estimates* the median intersection abscissas for each of the n^β randomly sampled dual lines. (All the other stages of the previous algorithm remain intact.) Because the algorithm returns, merely, an estimate of each of the medians, we cannot argue that it terminates in $O(1)$ stages in the expected case. Instead, we show that the expected number of stages is $O(\log n)$, yielding, thereby, an overall $O(n \log^2 n)$ expected running time. Empirical evidence of the algorithm's performance on numerous inputs from many distributions suggests, however, that for many realistic data sets, the estimated medians are close enough to the true medians, such that, for practical purposes, the algorithm does terminate in $O(1)$ stages, and hence runs in $O(n \log n)$ total time.

We begin by considering two simple strategies for converging on the repeated median intersection abscissa. Recall that the algorithm maintains an interval $(\theta_{\text{lo}}, \theta_{\text{hi}}]$ that contains the desired intersection abscissa. The task of each stage is to contract this interval. We consider two simple strategies for doing this.

The first strategy is to apply a simple randomized binary search for the repeated median intersection abscissa. It is easy to modify the counting procedure described in Section 2.1 to sample a single intersection abscissa θ_{mid} randomly from the interval. As in step (2d) of the algorithm in Section 2, we can determine the subinterval, $(\theta_{\text{lo}}, \theta_{\text{mid}}]$ or $(\theta_{\text{mid}}, \theta_{\text{hi}}]$, that contains the desired RM intersection abscissa, and then contract appropriately. Clearly after each stage we expect to eliminate a constant fraction of the remaining intersection abscissas from further consideration. Since the initial number of abscissas is $O(n^2)$, this randomized binary search terminates in $O(\log n)$ stages with high probability. Since each stage can be performed in $O(n \log n)$ time, the total running time is $O(n \log^2 n)$.

The shortcoming of this strategy is that it requires at least $\Omega(\log n)$ stages to terminate in the expected case. We introduce a second strategy that appears to converge much more rapidly in practice. This second strategy makes use of the observation (made in the previous section) that it is often easy to convert counting algorithms into sampling algorithms. In particular, this can be applied to the inversion counting procedure presented in Section 2.1 to generate a random subset of intersections on each of the n^β sampled dual lines. For this strategy, we set $\beta = \frac{1}{2}$, so that $O(\sqrt{n})$ dual lines are sampled. For each such line, we invoke the sampling procedure described below to return $O(\sqrt{n})$ randomly sampled intersection abscissas that lie in the interval $(\theta_{\text{lo}}, \theta_{\text{hi}}]$. Hence, the total number of sampled points is $O(n)$.

Let p_i be a point whose dual line, l_i, was sampled. Recall that L_i is the count of intersection abscissas that lie to the left of the interval, and C_i is the count of intersection abscissas that lie within the interval. Since the dual line of p_i was sampled, it is a candidate, and hence its median intersection abscissa lies within the current interval, $(\theta_{\text{lo}}, \theta_{\text{hi}}]$. Let $k_i = \lceil (n-1)/2 \rceil - L_i$ be the rank of θ_i within this interval. For each sampled intersection abscissa on l_i, the probability is k_i / C_i that it is less than or equal to θ_i. Thus the expected number of sampled intersection abscissas that are less than or equal to θ_i is roughly $\sqrt{n} k_i / C_i$. We select the sampled intersection abscissa whose rank is $\lceil \sqrt{n} k_i / C_i \rceil$ to be the estimate of the median intersection abscissa of the line l_i. Since

selection can be performed in linear time, the running time of this phase of the algorithm, i.e., after sampling has been completed, is $O(\sqrt{n}\sqrt{n}) = O(n)$.

The sampling itself can be performed in $O(n \log n)$ time by adapting the inversion counting algorithm, similarly to the way that was described in the previous section. Recall from the remarks of Section 2.1 that the set of intersection abscissas is in one-to-one correspondence with the set of inversions in a given permutation. Also recall that inversions are counted by a modification of mergesort. When two lists are merged in this algorithm, each element in one list implicitly discovers the subset of elements in the other list with which it generates an inversion. Note that this range comprises a contiguous sublist of the other list, and each element is involved in $O(\log n)$ such merges. In this way we can partition all the inversions involving any one element into a collection of $O(\log n)$ disjoint subsets. Using the same trick that we applied in Section 2.2, we can sample from these subsets in $O(\log n)$ time. The algorithm is described in detail in [24]. Since we are sampling $O(n)$ items altogether, it follows that the total running time of this sampling procedure is $O(n \log n)$.

Because the choice of the contracted interval for the second strategy is based on estimates of the lines' median intersection abscissas, pathological situations can be imagined in which the estimates are so skewed that the algorithm's rate of convergence is affected. Intuitively, this is similar to the phenomenon that interpolation search suffers from, in which highly skewed distributions can slow the algorithm's convergence rate arbitrarily. Thus, the algorithm operates by combining both strategies. The second (median estimation) strategy is applied by default. If at any stage the number of intersection abscissas fails to decrease by at least a modest constant factor, then the algorithm switches to the first (binary search) strategy for the next stage. Our own practical experience with the algorithm, on a number of distributions, has indicated that the estimated confidence interval traps the repeated median the vast majority of the time, and this extra generality is not really needed. However, it allows us to make the following claim.

LEMMA 3.1. *The expected running time of the algorithm based on estimating intersection abscissas is* $O(n \log^2 n)$.

4. Experimental Results. We implemented the algorithm described in the previous section and ran a number of experiments to study its performance on realistic input distributions. The experiment considered values of n ranging from 64 to 40,000, and 100 different data instances for each value of n. In each case, a set of n points associated with one of the following data distribution types was generated inside a unit square (see [24] for further details):

(1) Linear plus Gaussian perturbation.
(2) Linear plus one-sided (Weibull) perturbation.
(3) Bimodal plus Gaussian perturbation, i.e., a set of n points, n_1 of which are associated with one specified line, and $n_2 = n - n_1$ of which are associated with another specified line.
(4) Circular, i.e., points distributed along an arc of a circle.
(5) Uniform distribution in the unit square.

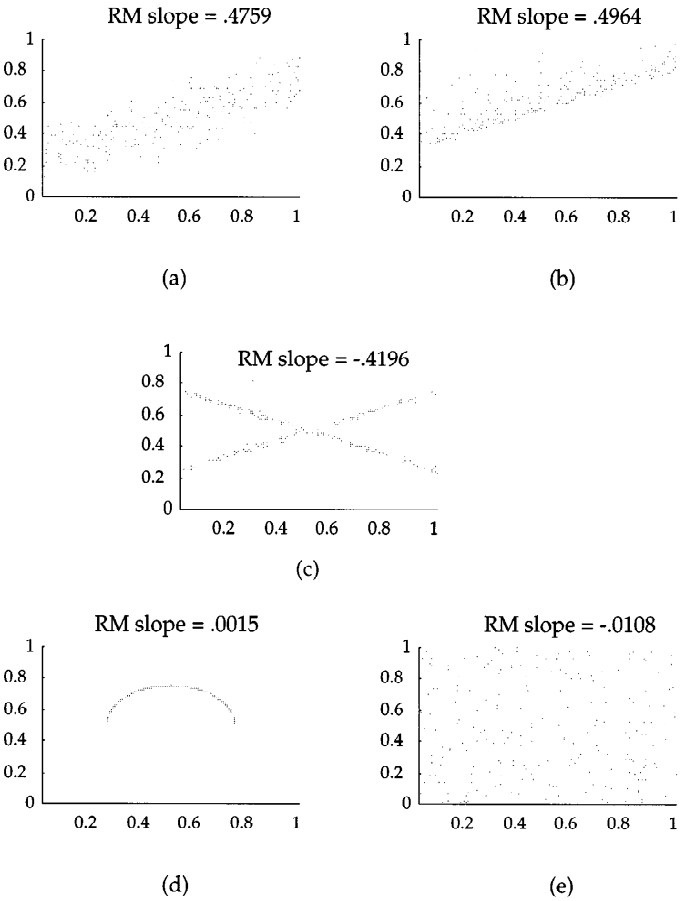

Fig. 2. Examples of experimental data sets ($n = 225$) for the following distribution types: (a) linear plus Gaussian; (b) linear plus Weibull; (c) bimodal plus Gaussian; (d) circular; (e) uniform in a unit square.

For each value of n, we ran our algorithm on 20 instances for each of the above types. The data instances generated correspond to lines of various slopes (and intercepts) and to various degrees of perturbation. Examples of various data instances for $n = 225$ are shown in Figures 2(a)–(e), depicting data instances of the distribution types (1)–(5), respectively. (The slopes of the linear sources of Figures 2(a)–(c) are either 0.5 or -0.5.) The corresponding RM values given as computed by the algorithm are shown.

For each data instance, we invoked our algorithm to find the RM estimate, and recorded the running times and the values of other parameters of interest (e.g., the number of iterations per each run, the number of times the algorithm has failed to capture the desired intersection abscissa in a contracted subinterval, etc.). For comparison, we have also recorded the running times of the $O(n^2)$ brute-force algorithm. In the vast majority of the data instances run, our algorithm terminated within *four* iterations for n values ranging from 64 to 40,000 (and, furthermore, the principal deviations from this rule were only for

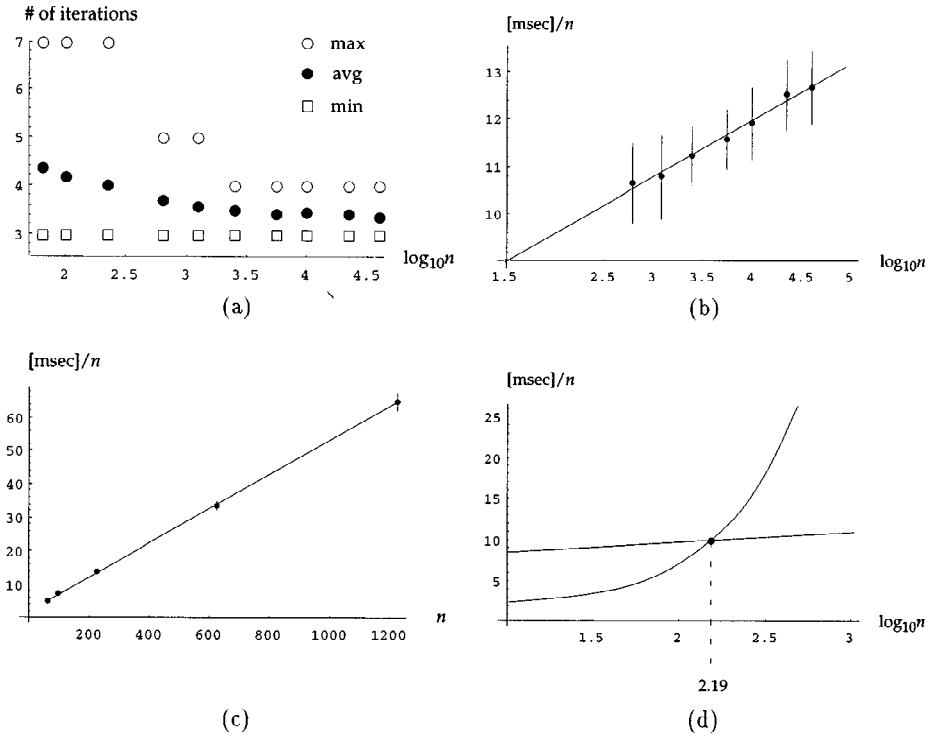

Fig. 3. Execution time statistics: (a) number of iterations observed versus $\log_{10} n$; (b) normalized running time versus $\log_{10} n$; (c) normalized brute-force running time versus n; (d) superimposing (b) + (c). The efficient algorithm is superior for $n > 155$.

small values of n, because various probabilistic arguments apply only to sufficiently large n). See Figure 3(a). Since each iteration takes $O(n \log n)$ time, this justifies empirically our claim that the algorithm runs essentially in (expected) $O(n \log n)$ time. Letting $T_{\mathrm{I}}(n)$ and $T_{\mathrm{B}}(n)$ denote, respectively, the running times (in milliseconds) of our inversion counting algorithm and the brute-force algorithm, we used least squares to derive the following linear models: $T_{\mathrm{I}}(n)/n = 1.2 \log_{10} n + 7.2$, and $T_{\mathrm{B}}(n)/n = 0.05n + 1.9$. The running times have been scaled down by a factor of n to reduce the residuals for large n. The results are summarized in Figures 3(b)–(d). Figure 3(b) shows $T_{\mathrm{I}}(n)/n$, Figure 3(c) shows $T_{\mathrm{B}}(n)/n$, and Figure 3(d) shows both fits superimposed. We conclude from these models that, for $n > 10^{2.19} \approx 155$, our algorithm performs faster than the brute-force algorithm. (It should be noted that the experiments were carried out using floating-point arithmetic, and that the running times reported might have been influenced by this fact.)

To investigate the probabilistic behavior of our algorithm, we also recorded the number of times the contraction stage has failed to trap the RM intersection abscissa, i.e., the frequency in which the RM intersection abscissa was not contained in the contracted subinterval, $(\theta'_{\mathrm{lo}}, \theta'_{\mathrm{hi}}]$. (Although the algorithm may take longer to run in such cases, step (2e) ensures its Las Vegas nature, i.e., the algorithm always returns a correct computational result.) With the choice of parameters made, it was observed that the

RM intersection abscissa was trapped successfully for over 99% of the cases, i.e., the algorithm exhibits an efficient probabilistic performance.

5. Conclusions. In this paper we presented two algorithms for the 50% breakdown point repeated median (RM) line estimator. First, an $O(n \log n)$ (expected) running time algorithm was provided. (This is the best theoretical result known.) An alternative, simpler variant of the above algorithm (which runs in $O(n \log^2 n)$ expected time) was also suggested and pursued. The characteristic features of the two algorithms are summarized as follows:

- Both algorithms are randomized in a Las Vegas sense, i.e., their (expected) running time complexities occur with high probability, and they always return the correct computational result (within the machine's precision).
- Both algorithms are space optimal, i.e., they require $O(n)$ storage.
- The algorithms are extendible to higher dimensions. Specifically, we have shown that a d-dimensional RM (hyperplane) estimator can be computed, with high probability, in (expected) $O(n^{d-1} \log n)$ ($O(n^{d-1} \log^2 n)$) time and $O(n)$ space, for fixed d [25].
- In principle, both algorithms are implementable. However, while the first algorithm relies on rather sophisticated data structures, the second algorithm is based, merely, on simple modifications of mergesort. We believe that the latter algorithm achieves the best combination of both asymptotic and practical efficiency among *exact* algorithms for 50% breakdown point estimators.

  Furthermore, empirical results for a large number of data sets strongly suggest that the (expected) running time of the second algorithm is essentially $O(n \log n)$, and that the constants of proportionality (hidden by the asymptotic notation) are relatively small.

  Finally, due to this practical algorithm, statistical properties of the RM estimator (e.g., asymptotic statistical efficiency) can now be studied and verified empirically for large values of n [28].

A few issues may be considered for future research. One immediate question is whether a deterministic $O(n \log n)$-time algorithm can be found for the RM line estimator (e.g., by derandomization of the algorithm provided). Another question regards the choice of the repeated median slope as a line estimator. Although the (repeated) median slope is preserved under translation and scaling, it is not the case that the (repeated) median slope of a rotated set of points is equal to the rotated (repeated) median slope (since, for example, when points are nearly vertical we may find two clusters of slopes at $+\infty$ and $-\infty$). Can the techniques of this paper be extended to estimators which are invariant under rotation?

References

[1] H. Brönnimann and B. Chazelle, Optimal slope selection via cuttings, in *Proceedings of the Sixth Canadian Conference on Computational Geometry*, Saskatton, Saskatchewan, August 2–6, 1994, pp. 99–103.

[2] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir (1993), Diameter, width, closest line pair, and parametric searching, *Discrete & Computational Geometry*, **10**, 183–196.
[3] H. Chernoff (1952), A measure of asymptotic efficiency for test of a hypothesis based on the sum of observations, *Annals of Mathematical Statistics*, **23**, 493–507.
[4] R. Cole (1987), Slowing down sorting networks to obtain faster sorting algorithms, *Journal of the ACM*, **34**, 200–208.
[5] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi (1989), An optimal-time algorithm for slope selection, *SIAM Journal on Computing*, **18**, 792–810.
[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest (1990), *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, and McGraw-Hill, New York.
[7] M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu (1992), A randomized algorithm for slope selection, *International Journal of Computational Geometry and Applications*, **2**, 1–27.
[8] D. L. Donoho and P. J. Huber (1983), The notion of breakdown point, in *A Festschrift for Erich L. Lehman*, edited by P. J. Bickel, K. Doksun, and J. L. Hodges, Jr., Wadsworth, Belmont, California, pp. 157–184.
[9] H. Edelsbrunner and D. L. Souvaine (1990), Computing median-of-squares regression lines and guided topological sweep, *Journal of the American Statistical Association*, **85**, 115–119.
[10] H. Edelsbrunner and E. Welzl (1986), Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Information Processing Letters*, **23**, 289–293.
[11] P. Erdös and J. Spencer (1974), *Probabilistic Methods in Combinatorics*, Academic Press, New York.
[12] W. Feller (1950), *An Introduction to Probability Theory and its Applications*, Vol. 1, Wiley, New York.
[13] R. Floyd and R. Rivest (1975), Expected time bounds for selection, *Communications of the ACM*, **8**, 165–172.
[14] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel (1986), *Robust Statistics*: *The Approach Based on Influence Functions*, Wiley, New York.
[15] D. Haussler and E. Welzl (1987), ε-nets and simplex range queries, *Discrete & Computational Geometry*, **2**, 127–151.
[16] C. A. R. Hoare (1970), Proof of a program: FIND, *Communications of the ACM*, **13**, 39–45.
[17] P. J. Huber (1981), *Robust Statistics*, Wiley, New York.
[18] M. Katz and M. Sharir (1993), Optimal slope selection via expanders, *Information Processing Letters*, **47**, 115–122.
[19] D. E. Knuth (1973), *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, Massachusetts.
[20] J. Matoušek (1991), Randomized optimal algorithm for slope selection, *Information Processing Letters*, **39**, 183–187.
[21] J. Matoušek (1992), Efficient partition trees, *Discrete & Computational Geometry*, **8**, 315–334.
[22] J. Matoušek, D. M. Mount, and N. S. Netanyahu (1993), Efficient randomized algorithms for the repeated median line estimator, in *Proceedings of the Fourth Annual ACM–SIAM Symposium on Discrete Algorithms*, Austin, Texas, January 25–27, 1993, pp. 74–82.
[23] N. Megiddo (1983), Applying parallel computation algorithms in the design of serial algorithms, *Journal of the ACM*, **30**, 852–865.
[24] D. M. Mount and N. S. Netanyahu (1991), Computationally efficient algorithms for a highly robust line estimator, Technical Report CS-TR-2816, University of Maryland, College Park, Maryland.
[25] D. M. Mount and N. S. Netanyahu (1994), Computationally efficient algorithms for high-dimensional robust estimators, *CVGIP*: *Graphical Models and Image Processing*, **56**, 289–303.
[26] P. J. Rousseeuw (1984), Least median-of-squares regression, *Journal of the American Statistical Association*, **79**, 871–880.
[27] P. J. Rousseeuw and A. M. Leroy (1987), *Robust Regression and Outlier Detection*, Wiley, New York.
[28] P. J. Rousseeuw, N. S. Netanyahu, and D. M. Mount (1993), New statistical and computational results on the repeated median line, in *New Directions in Statistical Data Analysis and Robustness*, edited by S. Morgenthaler, E. Ronchetti, and W. A. Stahel, Birkhäuser-Verlag, Basel, pp. 177–194.
[29] P. K. Sen (1968), Estimates of the regression coefficients based on Kendall's tau, *Journal of the American Statistical Association*, **63**, 1379–1389.
[30] L. Shafer and W. L. Steiger (1993), Randomizing optimal geometric algorithms, in *Proceedings of*

the Fifth Canadian Conference on Computational Geometry, Waterloo, Ontario, August 5–9, 1993, pp. 133–138.

[31] A. F. Siegel (1982), Robust regression using repeated medians, *Biometrika*, **69**, 242–244.

[32] D. L. Souvaine and J. M. Steele (1987), Efficient time and space algorithms for least median of squares regression, *Journal of the American Statistical Association*, **82**, 794–801.

[33] A. Stein and M. Werman (1992), Finding the repeated median regression line, in *Proceedings of the Third Annual Symposium on Discrete Algorithms*, Orlando, Florida, January 1992, pp. 409–413.

[34] H. Theil (1950), A rank-invariant method of linear and polynomial regression analysis (Parts 1–3), *Nederlandse Akademie Wetenchappen Series A*, **53**, 386–392, 521–525, 1397–1412.

[35] D. E. Willard (1982), Polygon retrieval, *SIAM Journal on Computing*, **11**, 149–165.